企业CORBA中间件的性能分析及优化方法

费洪晓,欧阳伟
(中南大学 信息科学与工程学院,湖南 长沙 410083)

摘 要:公共对象请求代理体系结构(CORBA)已在商业的分布式计算环境中得到广泛应用,因为它允许分布式应用程序进行交互,并且具有操作系统、网络协议、语言无关性,但是在高负荷通信的情况下,CORBA在处理并发请求时的性能比传统的socket协议逊色不少。文中旨在分析产生性能问题的主要因素并通过改进通用对象请求代理间协议(GIOP)以达到提高性能的目的。

关键词:CORBA;性能;改进;企业
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)03-0012-03

Performance Analysis and Improvement of Enterprise CORBA

FEI Hong-xiao, OUYANG Wei
(School of Information Science and Engineering, Central South University, Changsha 410083, China)

Abstract: Common object request broker architecture (CORBA) has a wide use in the commercial distributed computer environment, for it allows an interactive between distributed applications, and has irrelative with OS, network protocol and program language. However, CORBA has a less performance than socket protocol while dealing with subsequent request. Analyze the factors affecting the performance of enterprise CORBA and improve the performance presentation through the general Inter-ORB protocol.

Key words: CORBA; performance; improvement; enterprise

0 引 言

公共对象请求代理体系结构(CORBA)是对象管理。近年来公共对象请求代理体系结构(CORBA)已在商业的分布式计算环境中得到广泛应用,因为它允许分布式环境下的应用程序通过传递对象进行交互而不受操作系统、网络协议、程序语言的限制,CORBA的本质是透明化的C/S模式[1,2]。

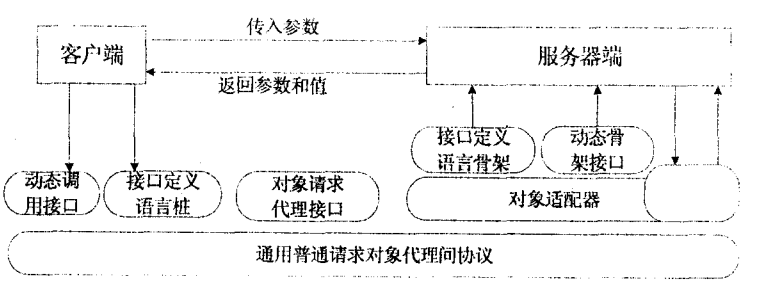

图1 CORBA系统结构图

在企业的应用中,客户端即查询界面,服务器端产生查询功能语句对数据层进行数据调用,客户端透明地通过向CORBA发出对象请求,CORBA将对象请求传递给服务器端,由服务器端完成对象功能的实现,从数据库中查询所需的结果并返回,如图1所示。

1 企业系统架构介绍及CORBA性能分析

在中国移动某集团的BOSS(运营支撑计费系统)中采用CORBA中间件处理手机用户的话费查询及其他业务办理功能,但是在营业前台大量用户并发进行话费查询、业务办理时,CORBA中间件在处理对象请求时出现性能降低的情况。调用CORBA服务时对客户端请求进行了封装,在对传输的数据结构进行编码解码的过程中将耗费大量的CPU、内存和I/O资源[3],文中将参照GIOP协议对CORBA数据结构的封装进行优化,从而达到提高性能的目的。

CORBA的核心为ORB(对象请求代理),它将客户端程序与它要使用的对象连接起来,发送服务请求的客户端只要知道实现其功能的对象的名字和对象接口的调用方法即可,ORB负责对象定位、对客户端请求的路由传递。

收稿日期:2008-07-13
基金项目:国家自然科学基金(60173041);湖南省自然科学基金(05JJY30119);湖南省科技计划项目(2006JT1040)
作者简介:费洪晓(1970-),男,副教授,主要研究方向为软件工程、网络管理、网络安全。

ORB 传递对象采用的协议是通用 ORB 间协议(General Inter ORB Protocol, GIOP),GIOP 协议内容包括:公共数据表示(Common Data Representation, CDR)、GIOP 消息格式、GIOP 消息传输、IIOP 协议(Internet ORB 间协议)。公共数据表示是将使用 OMG 接口定义语言(IDL)定义的数据类型映射为普通格式的网格化的消息表示。它描述了如何编码/解码操作中的数据,以便所有的服务器都可以抽取参数并调用远程过程,且数据交换不会产生多义性。编码/解码过程,实际上就是将某系统中特定的数据结构交换为公共数据表示的过程[2]。

公司总体结构从底层向上依次如图 2 所示。

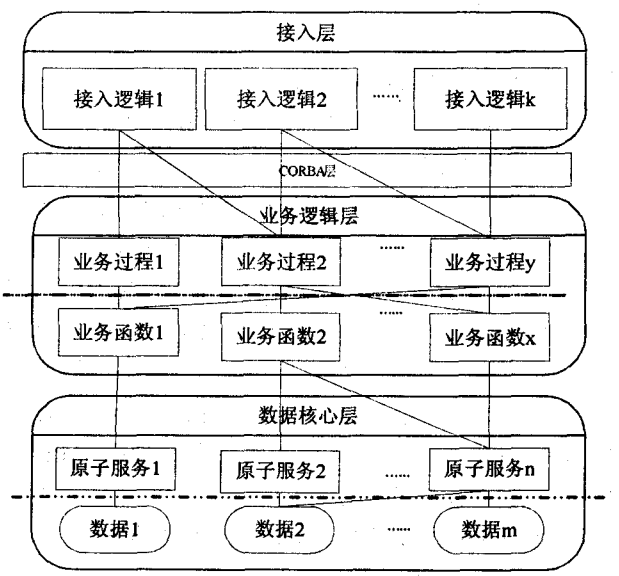

图 2　公司总体架构图

a.数据核心层是 BOSS 系统的基础,对业务逻辑层提供原子功能服务(查询);分为数据子层和服务子层,服务子层为逻辑层访问数据子层以统一规范的接口形式提供原子功能服务,将数据逻辑和业务逻辑分隔,增强了数据子层的安全性和一致性。该层采用 oracle 数据库存储数据。

b.业务逻辑层通过对若干原子服务功能的组合以满足不同的业务需求,分为业务函数与业务过程,业务过程由业务函数加上控制逻辑组合而成。该层采用 CORBA 软件 VisibrokerForC++ 编写实现功能的对象以响应客户端的调用,向 oracle 进行记录查询,一条记录对应一个对象。

c.接入层位于 BOSS 系统的外围,由接入逻辑构成,通过多种灵活方式与业务逻辑层相联,如 Internet 接入、营业服务终端接入、呼叫接入。该层使用 Java 语言,架构为 MVC 三层架构模型 struts。该层采用 CORBA 软件 VisibrokerForJava 编写客户端向业务逻辑层发送服务请求。

业务层向数据核心层提出请求并返回结果的速度很快,因为是底层直接操作数据库,很难有性能改善的空间。性能瓶颈主要发生在接入层将请求传递给业务逻辑层时。传统的集中式 C/S 模式采用 socket 协议传递请求,速度优于 CORBA 的 GIOP 协议,但是集中式 C/S 模式存在着客户端数量有限、无法满足客户随时随地进行业务办理的需求。随着分布式商业应用的兴起,CORBA 的出现很好地解决了这一问题,因为 CORBA 的平台与网络协议无关性,使客户可以通过营业厅、手机界面、网上营业厅、10086 等多种方式办理业务,提高了服务质量,不过在给客户提供方便的同时却带来了性能上的影响。客户端的请求发送给 CORBA 时需编码,为了方便业务上理解,公司对数据结构采用了封装。封装后的数据流传递至业务逻辑层后需要解码。业务逻辑层对数据库进行操作并返回结果时同样需要对结果进行编码/解码两步[4]。对大量请求的编码/解码过程是造成性能降低的一个重要原因。

2　优化 GIOP 层以提高性能

GIOP 协议可以映射到任何满足一个最小假设集的面向连接的传输协议上,它的目标是追求最广泛的可用性、简单性、可缩放性、低代价、通用性、体系结构的独立性。CDR(公共数据表示)转移语法是 GIOP 用字节流表示 OMG IDL 数据类型的格式。字节流通常与通过某种进程间通信机制或网络传输层发送到其他进程或机器上的一段内存缓冲相对应。

GIOP 定义了两种不同类别的字节流:消息和封装。消息是 GIOP 中信息交换的基本单位,封装是 OMG IDL 数据结构独立打包成的字节流。数据结构被封装后,此字节流就可表示为 OMG IDL 不透明的数据类型 sequence<octet>,随后这种数据类型可以被打包进消息体或另外的封装。

由于造成 CORBA 性能瓶颈的重要原因就是对封装的数据结构的编码和解码,文中拟提出以下两种方式对其进行一定的优化。

2.1　封装时采用基本数据类型边界对齐以减少填充字节

数据类型与对应的字节长度见表 1。

表 1　数据类型与对应的字节长度

数据类型	长度(字节)
Char, octet, Boolean, wchar	1
Short, unsigned short	2
String, Long, unsigned long, float, enumerated types	4
Long long, unsigned long long, double, long double	8
wchar	1,2,4

公司的 CORBA 数据结构封装采用的是按边界对

齐的,如 IDL 封装的数据结构文件中一个片段:

```
struct SVipRecord
  {
   long long m_llMarker;
   long long m_llDoneCode;
   short m_nActivityWay;
   string m_dateLeavetime;
   long long m_llAmount;
   double m_dTotalscore;
     ......
};
typedef sequence<SVipRecord> SVipRecordSeq;
```

边界对齐是为了减少 CPU 访问内存的次数。边界对齐时,任何一个基本类型的数据,其首字节在字节流中的下标必须是自身长度的整数倍。公司中服务器 CPU 为 64 位,当顺序从结构体 SVipRecord 中取数据时,取 long long m_llAmount 需要取两次,取 double m_dTotalscore 也需要两次,若将变量按字节数从大到小重新排列成如下形状,变量都只需取一次即可。

```
struct SVipRecord
{
   long long m_llMarker;
   long long m_llDoneCode;
   long long m_llAmount;
   double m_dTotalscore;
   string m_dateLeavetime;
   short m_nActivityWay;
   ......
   };
```

在后面的实验中,采用这种方法能使客户端做并发操作返回结果时间减少 15%。

2.2 减少封装的使用

GIOP 规定在三种情况下必须采用封装:复杂类型的 TypeCodes,在 IOR(可互操作的对象引用)中的 IIOP Profiles,和面向特定服务的上下文。封装编码和解码加重了内存和 CPU 的负担[5]。

```
struct SOrigBusiRecord
{
   long long m_llMarker;
   long long m_llDoneCode;
     ......
   string m_strRespCode;
};
   Typedef sequence<SOrigBusiRecord> SOrigBusiRecordSeq;
```

如上所示,SOrigBusiRecordSeq 是 IDL 中一个被封装的一长串属性列表的数据结构,它在调用 get_busiLogBySpecialCondition 方法时被当作一个参数(SOrigBusiRecordSeq_out)传递。

```
CORBA::Short IBossBusiOrig::get_busiLogBySpecialCondition(
    const char * strCondition, SOrigBusiRecordSeq_out seqOrig-
BusiRecord,
     SErrorMsg_out sErrMsg)
  {
    ......
  }
```

在传递 SOrigBusiRecordSeq_out 时可能只是想得到其中一个属性值,由于采用了封装,不得不把 SOrigBusiRecord 的所有属性值编码/解码并重传一遍,增加了编码和解码的负担,在实验中采取精确传递某个属性值,有效减少网络流量和编码解码,从而减少 CPU 和内存的消耗,提高系统性能。

3 对比实验结果与分析

服务器端后台采用 IBM 的 AIX5L 环境,CORBA 后台用 BES VISIBROKER 布署。在短信厅程序测试环境 c_so_busi 短信业务受理类中编写两个函数 Fun100001 和 Fun100002,分别采用以下两种方法做对比测试。在数据库短信接收表(接收手机发送的查询短信)中手工插入 100 条记录模拟大量用户通过短信厅做查询,当用户发送'HF'到 10086 时调用 Fun100001,用户发送'HFCX'时调用 Fun100002,查看 CORBA 后台日志,对比从传入查询参数到调用 CORBA 后得到查询结果的时间差,得到以下结果。

3.1 基本数据类型边界对齐以减少填充字节

Fun100001 调用的 IDL 未经边界对齐,Fun100002 调用的 IDL 经过边界对齐优化。对比结果见表 2(100 个查询)。

表 2 边界对齐与性能比较结果

	耗费时间(秒)	效率提高%
未边界对齐	2.35	
边界对齐	2.11	10.2%

3.2 减少封装的使用

Fun100001 传入整个封装的 IDL,Fun100002 仅传递用户号码,花费的时间对比见表 3(100 个查询)。

表 3 封装与性能比较结果

	耗费时间(秒)	效率提高%
传整个 IDL	2.33	
仅传属性值	1.95	16.3%

4 结束语

常用以上两种方法都能对客户端查询效率起到一定的提高,企业采用传递整个 IDL 是为了便于程序的维护,但是以牺牲效率为代价。文中对企业的 CORBA

(下转第 18 页)

⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮

R_{n*m}: IF x is A_m AND y is B_n THEN z is C_6;

则建立的规则表,如表 2 所示。

表 2 模糊规则表

x \ y	B_1	…	B_n
A_1	C_1		C_p
A_2	C_2	…	C_5
…	…	…	…
A_m	C_4	…	C_6

在完成规则表之后,对于每一条规则,需要先求出其相应的模糊关系,例如,对于规则 R_1:

IF x is A_1 AND y is B_1 THEN z is C_1

模糊蕴涵关系为:

$((x \text{ is } A_1) \wedge (y \text{ is } B_1)) \rightarrow (z \text{ is } C_1)$

那么对应的模糊关系 R_1 为:

$$R_1 = A_1 \times B_1 \times C_1$$

同理,可以为其他所有的规则求出其模糊关系 $R_2, R_3, \cdots, R_{n*m}$,最后,可求得与模糊控制规则表相对应的模糊关系矩阵,即:

$$\boldsymbol{R} = R_1 \vee R_2 \vee \cdots \vee R_{n*m}$$

在求出模糊关系矩阵 $\boldsymbol{R}$ 后,对于任意的输入 $A^* B^*$,则输出 C^* 为:

$$C^* = (A^* \times B^*) \cdot \boldsymbol{R}$$

由上述矩阵 $\boldsymbol{R}$ 的计算过程可以看到,求解 $\boldsymbol{R}$ 阵的工作量是相当大的,但大都是重复的并行计算工作,用软件计算需要花费大量的时间,而 FPGA 体系结构通常由相对简单的逻辑单元阵列和大量的寄存器组成,特别适用于实现并行计算。

4) 反模糊化模块。

模糊推理的输出结果是一个模糊集,而模糊控制器的输出必须是一个确定的数值,用于实际控制。这样就需要将推理结果的反模糊化,最常用的是采用重心法,也称力矩法,其本质为加权平均法[7],对于论域为离散的情况有:

$$c = \frac{\sum C_i \mu(C_i)}{\sum \mu(C_i)}$$

4 结束语

针对模糊控制应用领域广泛的特点,采用 FPGA 技术设计了一个通用模糊控制器的模型,即克服了传统的纯电路硬件无重用性的缺陷,也弥补了软件控制器在实时性和稳定性方面的不足。在具体模块的设计方面,以目前的主流算法为例,详细介绍了各个模块的设计细节,使得在保持整体结构的不变的前提下,只需要选择合适的核心算法,便可以设计出针对某一领域的模糊控制器,从而满足了模糊控制器灵活性和可定制的特点。同时,FPGA 设计以强大 EDA 技术和硬件描述语言为平台,具有周期短、成本低的特点,有着良好的实用价值和乐观的应用前景。

参考文献:

[1] 石 磊,陈亚娜. 模糊控制器的设计研究[J]. 机电产品开发与创新,2007(6):108-110.

[2] 程耀林. FPGA 的系统设计方法解析[J]. 微型电脑应用,2007(1):48-51.

[3] 肖兵梁,志 坚. 两款通用模糊控制器的设计与实现[J]. 自动化与仪器仪表,2005(1):13-16.

[4] 贺今朝. 一种基于 FPGA 的模糊控制器的研究[D]. 大连:大连理工大学,2002.

[5] 沃尔夫 W. 基于 FPGA 的系统设计(英文版)[M]. 北京:机械工业出版社,2006.

[6] 褚振勇. FPGA 设计及应用[M]. 第 2 版. 西安:西安电子科技大学出版社,2006.

[7] 诸 静,闫敬文. 模糊控制原理与应用[M]. 第 2 版. 北京:机械工业出版社,2005.

(上接第 14 页)

中间件在实际使用中遇到的问题进行分析,从编码/解码方面做了一些改进。目前已有针对性能敏感的实时应用的 CORBA 产品,其中 TAO 的应用最为广泛,它基于底层的中间件 ACE(Adaptive Communication Environment)。ACE 实现了并发和分布式功能,目的是满足实时应用的可确定性和服务质量的需求。

随着这种满足各种特定需求的中间件产品的不断出现,它在分布式应用中的前景将会更加广阔。

参考文献:

[1] 李 方,张 虹,GIOP 协议和 CORBA 的性能优化[J]. 微计算机信息(管控一体化),2006,22(7):157-159.

[2] 韦乐平,薛君敖,孟洛明. CORBA 系统结构、原理与规范[M]. 北京:电子工业出版社,2000.

[3] 鲁 波,邹 华,王 柏. CORBA 性能分析及优化[J]. 计算机工程,2001,27(9):3-4.

[4] Tao W, Majumdar S. Application Level Performance Optimizations for Systems[C]//Proc. International Workshop on Software and Performance (WOSP02). Rome: [s. n.], 2002: 95-103.

[5] Ahmad I, Majumdar S. Achieving High Performance in CORBA-Based Systems with Limited Heterogeneity[R]. Ottawa, Canada: Dept. of Systems and Computer Engineering, Carleton University, 1999.